hhu.

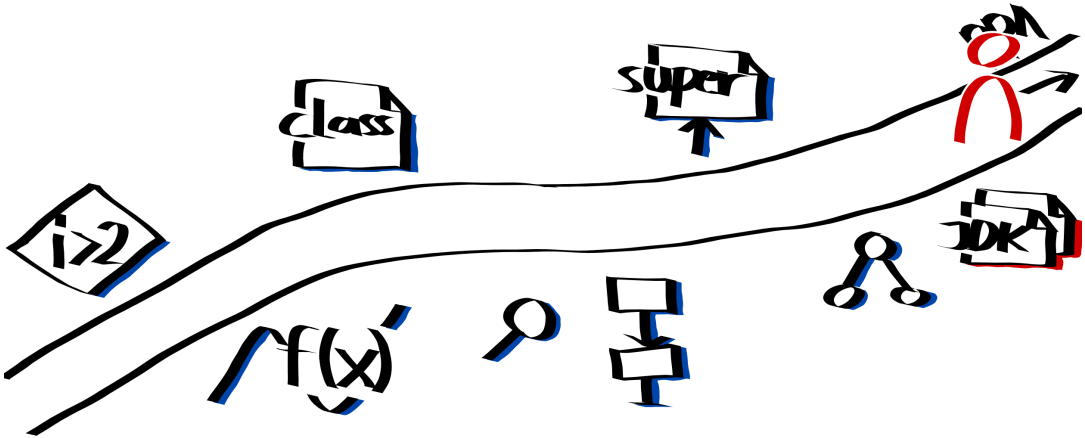
Kapitel 8: Packages, Frameworks & Co.

Reguläre Ausdrücke & Automatisches Testen

Wintersemester 2023/24

Stand 3. August 2023, 15:38 Uhr

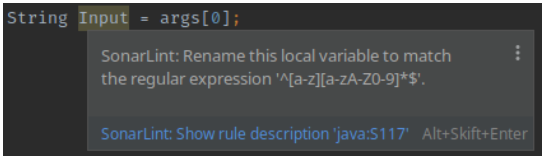
Wo stehen wir gerade?



Vorlesungsfeedback

Reguläre Ausdrücke[1]

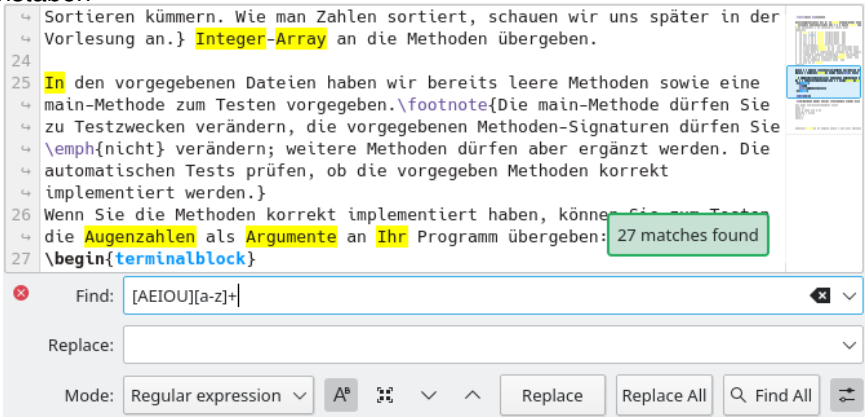
- Motivation: Beschreibung einer Menge von Zeichenketten
- Anwendungen:
  - Filterkriterien in Textsuche
  - Definition formaler Sprachen
    →TheoInfo →Compilerbau
  - Automatisches Suchen & Ersetzen
  - Telefonnummern in Textnachrichten erkennen



[1] Regular Expressions, RegExp, Regex

Beispiel: Suche im Texteditor

Finde alle Strings, die mit großem Vokal beginnen, gefolgt von beliebig vielen Kleinbuchstaben



Terminal-Programme: `grep`, `sed`

Heinrich Heine Universität Düsseldorf

```
% grep --color " [A-Z]*[ -]" register.txt
Im PC-Register ist üblicherweise eine Adresse, die innerhalb der Method Area liegt, gespeichert. Im
LV-Register ist üblicherweise die Wortadresse des Anfangs des Stack-Bereiches, bei dem die der aktue
llen Funktion gespeichert sind, hinterlegt. Im SP-Register speichert man üblicherweise die Wortadres
se des letzten Wertes auf dem Stack.
Das Register MAR speichert die Hauptspeicheradresse, von der gelesen bzw. an die geschrieben werden
soll (wortweise Adressierung). Das Register MDR speichert den Wert, der in den Hauptspeicher geschri
eben werden soll (wr) bzw. den Wert, der aus dem Hauptspeicher gelesen worden ist (rd).

% sed "s/ [A-Z]*[ -]/ ____/g" register.txt
Im ____Register ist üblicherweise eine Adresse, die innerhalb der Method Area liegt, gespeichert. Im
 ____Register ist üblicherweise die Wortadresse des Anfangs des Stack-Bereiches, bei dem die der akt
uellen Funktion gespeichert sind, hinterlegt. Im ____Register speichert man üblicherweise die Wortad
resse des letzten Wertes auf dem Stack.

Das Register ____speichert die Hauptspeicheradresse, von der gelesen bzw. an die geschrieben werden
soll (wortweise Adressierung). Das Register ____speichert den Wert, der in den Hauptspeicher geschri
eben werden soll (wr) bzw. den Wert, der aus dem Hauptspeicher gelesen worden ist (rd).
```

Grundlegende Konstruktionsregeln

Heinrich Heine Universität Düsseldorf

- Elementare Zeichen sind reguläre Ausdrücke (stehen für sich selbst): `0`, `1`, `A`, `a`, ...
- Auswahl:
  - `[aeiou]` matcht einen Vokal
  - `[a-z]` matcht einen Kleinbuchstaben von a bis z
  - `[1-9]` matcht eine Ziffer von 1 bis 9
  - `[A-Ca-c]` matcht A, B, C, a, b, oder c

Seien p und q reguläre Ausdrücke:

- pq: matcht p gefolgt von q
- $p*$: matcht p beliebig oft
- $p+$: matcht p mind. einmal

Welche Strings von `[A-Z]+[a-z]*` exakt gematch?

Heinrich Heine
Universität Düsseldorf

1. HAL1O
2. HALLO
3. Hallo
4. HALlo
5. hallo

Welche Strings von `[A-Z]+[a-z]*` exakt gematch?

Heinrich Heine
Universität Düsseldorf

1. HAL1O
2. **HALLO**
3. **Hallo**
4. **HALlo**
5. hallo

Reguläre Ausdrücke konstruieren

Heinrich Heine Universität Düsseldorf

Welcher Ausdruck beschreibt einen String, der

- mit exakt einem A, B, L oder K beginnt
- gefolgt von mindestens einer Ziffer, wobei die erste Ziffer keine 0 ist

(also z. B. A45, B1, L381, aber nicht A04, l3, A)

1. `[A-Z][0-9]+`
2. `[abkl][0-9]+`
3. `[ABLK][1-9][0-9]*`
4. `[ABLK][1-9][0-9][0-9]*`

Reguläre Ausdrücke konstruieren

Heinrich Heine Universität Düsseldorf

Welcher Ausdruck beschreibt einen String, der

- mit exakt einem A, B, L oder K beginnt
- gefolgt von mindestens einer Ziffer, wobei die erste Ziffer keine 0 ist

(also z. B. A45, B1, L381, aber nicht A04, l3, A)

1. `[A-Z][0-9]+`
2. `[abkl][0-9]+`
3. **`[ABLK][1-9][0-9]*`**
4. `[ABLK][1-9][0-9][0-9]*`

Mehr zu regulären Ausdrücken

Heinrich Heine Universität Düsseldorf

- Weitere Syntax:[2]
  - `.` matcht ein beliebiges Zeichen
  - `[^abc]` matcht ein Zeichen, das *nicht* a, b, c ist
  - `\w` matcht einen Buchstaben (auch Umlaute etc.), Ziffer oder Unterstrich
  - `p?`: match `p` höchstens einmal
  - `p|q`: matcht entweder `p` oder `q`
  - `p{n,m}`: matcht `p` mind. *n*-mal, max. *m*-mal
  - `(p)`: Capture-Group (zur Verwendung bei Suchen & Ersetzen oder zur Gruppierung)
  - `$`: matcht Zeilenende
  - ...
- Formal: Definition regulärer Sprachen →TheoInfo
  - Bemerkung: Reguläre Ausdrücke in der Praxis können mehr als die aus der theoretischen Informatik

[2] https://de.wikipedia.org/wiki/Regulärer_Ausdruck#Reguläre_Ausdrücke_in_der_Praxis

Reguläre Ausdrücke in Java

Heinrich Heine Universität Düsseldorf

```java
import java.util.regex.Pattern;

public class IsStreet {
    public static void main(String[] args) {
        String streetNamePattern = ("[ABKL][1-9][0-9]*");
        String input = args[0];
        boolean isStreetName = Pattern.matches(streetNamePattern, input);
        System.out.println(isStreetName);
    }
}
```

```
% java IsStreet A45
true
% java IsStreet B1
true
% java IsStreet b1
false
% java IsStreet K
false
```

- Klassen `Pattern` und `Matcher` können noch viel mehr (z. B. Extrahieren von Teilstrings)

Automatisches Testen

Heinrich Heine
Universität Düsseldorf

- Motivation: händisches Testen auf Dauer unpraktisch
  - kann leicht vergessen werden
  - ständige Prüfung aller Randfälle mühselig
  - schwierig automatisierbar
- Idee: Auflistung von Bedingungen, die korrekte Codeeinheit erfüllen soll
  - Codeeinheit: typischerweise Klasse

  → „Unit-Testing“

Erste Idee: `assert`s in main-Methode

Heinrich Heine Universität Düsseldorf

```java
public class Sum {

    public static int of(int a, int b) {
        return a + b;
    }

}
```

```java
public class SumAsserts {

    public static void main(String[] args) {
        assert Sum.of(3, 5) == 8;
        assert Sum.of(2000000000, 1000000000) == 3000000000L;
        assert Sum.of(-13, -49) == -62;
    }

}
```

Ausgabe

Heinrich Heine
Universität Düsseldorf

```
% java -ea SumAsserts
Exception in thread "main" java.lang.AssertionError
        at SumAsserts.main(SumAsserts.java:5)
```

Probleme:

- aus Ausgabe nicht erkennbar, was Fehler ist
- Abbruch nach erstem Fehler

Zweite Idee: Exceptions abfangen

```java
public class SumCatchAsserts {
    public static void main(String[] args) {
        try {
            assert Sum.of(3, 5) == 8;
        } catch(AssertionError e) {
            System.out.println("8 != " + Sum.of(3, 5));
        }
        try {
            assert Sum.of(2000000000, 1000000000) == 3000000000L;
        } catch(AssertionError e) {
            System.out.println("3000000000 != " + Sum.of(2000000000,
              ↪   1000000000));
        }
        try {
            assert Sum.of(-13, -49) == -62;
        } catch(AssertionError e) {
            System.out.println("-62 != " + Sum.of(-13, -49));
        }
    }
}
```

Ausgabe

```
% java -ea SumCatchAsserts
3000000000 != -1294967296
```

Probleme:

- relativ umständlicher Code
  - lenkt von eigentlichen Bedingungen ab
  - viel doppelt aufzuschreiben

Lösung: Bibliotheken für automatische Tests

Heinrich Heine Universität Düsseldorf

```java
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

public class SumTest {
    @Test
    public void addPositiveNumbers() {
        int result = Sum.of(3, 5);
        assertEquals(8, result);
    }

    @Test
    public void addVeryBigNumbers() {
        long result = Sum.of(2000000000, 1000000000);
        assertEquals(3000000000L, result);
    }

    @Test
    public void addNegativeNumbers() {
        int result = Sum.of(-13, -49);
        assertEquals(-62, result);
    }
}
```

Ausgabe

Heinrich Heine Universität Düsseldorf

Ausgabe der Bibliothek JUnit 5:

```
% javac -cp junit-platform-console-standalone-1.8.0-M1.jar:. SumTest.java
% java -jar junit-platform-console-standalone-1.8.0-M1.jar -cp . --scan-class-path
|
|- JUnit Jupiter pass
|  |- SumTest pass
|     |- addVeryBigNumbers() fail, expected: <3000000000> but was: <-1294967296>
|     |- addPositiveNumbers() pass
|     |- addNegativeNumbers() pass
|- JUnit Vintage pass
```

- alle Methoden mit `@Test` automatisch ausgeführt
- gute Fehlermeldungen geschenkt
- Praxis: auch andere Bibliotheken (z. B. AssertJ), automatische Testausführung (CI) →Propra

Aber: Der Aufruf ist ja total umständlich?

Aber: Der Aufruf ist ja total umständlich?

Heinrich Heine
Universität Düsseldorf

→ IDE und Build-Tools helfen weiter →Propra

- Im Hintergrund passiert weiterhin das Gleiche
  - jar-Datei mit Bibliothek benutzt
  - Classpath richtig gesetzt

! Software-Tests integraler Bestandteil professioneller Softwareentwicklung

- mehr Vertrauen in Korrektheit des Codes
- Schutz vor versehentlichen Verschlimmbesserungen
- Dokumentation von erwünschtem Verhalten

Testen Sie typische Randfälle

Heinrich Heine
Universität Düsseldorf

- Zahlen:

Testen Sie typische Randfälle

- Zahlen: 0, negative Zahlen, extrem große Werte, NaN, Infinity
- Referenzen:

Testen Sie typische Randfälle

- Zahlen: 0, negative Zahlen, extrem große Werte, NaN, Infinity
- Referenzen: null
- Datenstrukturen:

Testen Sie typische Randfälle

- Zahlen: 0, negative Zahlen, extrem große Werte, NaN, Infinity
- Referenzen: null
- Datenstrukturen: Anfang/Ende/Mitte/leer

> *A QA[3] engineer walks into a bar. Orders a beer. Orders 0 beers. Orders 99999999999 beers. Orders a lizard. Orders -1 beers. Orders a ueicbksjdhd.*
>
> *First real customer walks in and asks where the bathroom is. The bar bursts into flames, killing everyone.*

– Brenan Keller, Entwickler bei Snapchat

[3] Quality Assurance

Lernziele

Heinrich Heine
Universität Düsseldorf

Sie können am Ende der Woche ...

- **angeben**, ob ein gegebener String zu einem regulären Ausdruck passt.
- **begründen**, warum automatische Software-Tests sinnvoll sind.

Wichtige Begriffe

Heinrich Heine
Universität Düsseldorf

Regulärer Ausdruck [A-Za-z] [0-9]+
[123]* match Unit-Test

Fragestunde, Klausur

Heinrich Heine
Universität Düsseldorf

- Dienstag: Zulassungsinfos via Studiportal
- Dienstag: letzter Tag der Klausuranmeldung
  - ! nur mit Anmeldung ist mitschreiben möglich
  - falls Anmeldung über Portal nicht klappt: progra@cs.uni-duesseldorf.de
- Mittwoch: Fragestunde; Fragen im Vorlesungsfeedback hinterlassen
- Mittwoch: Infos zum Klausurablauf